Parallel Algorithm to find a Weighted Core of a Tree

S.P. Victor[1] C. Xavier[2] S. Arumugam[3]

ABSTRACT : Let T be a tree in which both the vertices and edges are assigned weights. For any two vertices u, v in $V(T)$, the weighted distance $wd(v,u)$ is defined to be the product of the weight of the vertex u and the sum of the weights of the edges of the unique v - u path in T. If $s \subseteq V(T)$ and $v \in V(T)$, $wd(v,s) = \min_{u \in s} wd(v,u)$. The *w-core* of T is a path P in T for which $wd(P)=\sum_{v \in V(T)} d(v,P) * w(v)$ is minimum. In this paper we present a parallel algorithm for finding the *w-core* of T and prove that the algorithm requires $O(\log^2 n)$ time using $O(n^3)$ processors on the *CREW PRAM*.

KEYWORDS :Parallel algorithm, Tree, Weighted tree, Core, Weighted core

1. INTRODUCTION

The centrality concepts in graphs such as centre, centroid and median are widely applied in facility location problems and social networks. Facility location problems arise from several real-life situations wherein we deal with the question of determining an optimal location for a facility in a graph.

The nature of the facility to be constructed could necessitate selecting a structure such as a path rather than just a point at which to locate the facility. Motivated by this observation Slater [6] extended the concept of eccentricity *e*, distance *d*, and branch weight *bw* for paths and paths P for which $e(P)$, $d(P)$ and $bw(P)$ are minimized are called *path centers, cores* and *spines* respectively. Minieka and Patel [3] have investigated the problem of finding a tree core with a specified length. Morgan and Slater [4] have proposed a linear algorithm for computing a core of a tree. Albacea [1] has presented a parallel algorithm for finding a core of a tree.

In all the earlier works [1, 3, 4, 6] it was assumed that the edges of the graph have weights, usually the distance. However in reality, the vertices may also have weights, for example, the population of the city represented by the vertex or the number of units of commodity to be transported from a vertex. In case of a network of computers, the weight of the vertex may be taken as the intensity of the data traffic initiated from the vertex. In this paper we design parallel algorithms to find the w-core in trees where both edges and vertices are assigned weights. For graph theoretic terminology, we refer to Harary [2] and Parthasarathy [5].

2. FINDING W-CORE IN PARALLEL

Let $T=(V,E)$ be a tree rooted at r, with weights on edges as well as on vertices. Such a tree is called a *weighted tree*. The weighted distance from a vertex v to a vertex u is defined by $wd(v,u)=d(v,u)*w(u)$, where $d(v,u)$ is the sum of the edge weights in the unique v - u path in T, and $w(u)$ is the weight of the vertex u. The weighted distance $wd(v,u)$ may be interpreted as the cost for transporting $w(u)$ units from the vertex u to the vertex v. We note that the weighted distance is not symmetric so that $wd(v,u) \neq$

[1] Reader in Computer Science, St. Xavier's College (Autonomous), Palayamkottai – 627 002, Tamil Nadu, India, E-mail : victorsp@rediffmail.com

[2] HCL Technologies, Chennai, India,

[3]Senior Professor (Research), Arulmigu Kalasalingam College of Engineering, Krishnankoil-626 190, Tamil Nadu, India.

$wd(u,v)$. The *weighted eccentricity (w-eccentricity)* of v is defined by $we(v)=max\ \{\ wd(v,u);\ u \in V(T)\ \}$.

For a subset S of $V(T)$, we define $wd(v,S) = \min_{u \in S} wd(v,u)$. The *w-core* of a tree T is defined to be a path P in T for which $wd(P)=\sum_{v \in V(T)} wd(v,P) * w(v)$ is minimum. A Parallel algorithm to find the w-core of a weighted tree is discussed in this paper.

Theorem 1. A tree T rooted at r can be re-rooted at a vertex v in $O(1)$ time using $O(n)$ processors on the *CREW PRAM*.

Proof. Let T be the tree rooted at r and v be an arbitrary vertex in V. Let S be the set of vertices in the path from v to r. Each vertex u in S except v has exactly one child in S and that child is taken as the parent of u in the re-rooted tree and v has no parent. The parent for each of the remaining vertex is the same as its parent in the original tree T. Such a tree T_v is the tree T re-rooted at v.

Elements of S can change their parents simultaneously in 1 unit time using $O(n)$ processors. Hence a tree T rooted at r can be re-rooted at v in $O(1)$ time using $O(n)$ processors.

Let $T=(V,E)$ be a weighted tree rooted at r. Let $P_{r,v}$ be the path from the root r to a vertex v. Let Q be a path, with r as its terminal vertex and intersecting $P_{r,v}$ only at r. Let $wd(Q)=\sum d(v,Q) * w(v)$. Then the *w-reduction(r,v)* is defined to be the total reduction in $wd(Q)$ when the path Q ending at r is extended along the path $P_{r,v}$. Thus w-*reduction(r, v)* $= \sum d(p(x),x) * wd(x);\ x \in P_{r,v} - \{r\}$, where $p(x)$ is the parent of x.

The weighted distance of a subtree T_v rooted at vertex v, is defined as follows:

$stwd(v) = \sum_{u \in V(Tv)} d(v,u) * w(u)$, if v is an internal vertex of T, and

$stwd(v) = w(v)$, for terminal vertices

Lemma 2. Let R be a path ending at r. The total *w-reduction* in $wd(R)$ when R is extended to include the vertices in $P_{r,v} - \{r\}$ is given by *w-reduction(r,v)*$=\sum d(p(x),x) * stwd(x);\ x \in P_{r,v} - \{r\}$, where $p(x)$ is the parent of x and $stwd(x)$ is the weighted distance of the subtree rooted at x.

Proof. Extending path R from r to an adjacent vertex $x \in P_{r,v} - \{r\}$ will decrease $wd(R)$ by $wd(r,x)*stwd(x)$. Repeating this process on the next vertex in $P_{r,v}$, until vertex v is reached will obviously decrease $wd(R)$ by $\sum d(p(x),x) * stwd(x)$.

Remark : The above lemma shows that *w-reduction(r,v)* is independent of the path Q used for computation. Hence for computational purpose we can choose any vertex x, and Q to be the path $P_{x,r}$ in T, having r as the only common vertex with $P_{r,v}$.

Lemma 3. Let C be a *w-core* of a weighted tree T. Let r be the root of T and $r \in C$. The *w-core* C extends from r to two vertices e_1 and e_2 such that *w-reduction(r,e1)*+*w-reduction(r,e2)*=max($\{$*w-reduction(r,x)*+*w-reduction(r,y)* $| x,y \in V\}$). Moreover, the paths $P_{r,e1}$ and $P_{r,e2}$ intersect only at r.

Proof. By the definition of a *w-core*, it follows that the *w-core* extends from r into two non-intersecting paths where $\sum wd(r,v);\ v \in V(T)$ is reduced the most.

Lemma 4. Let T be a weighted tree rooted at r. If *w-reduction(r,v)* is equal to the maximum possible value, then v is in a *w-core* of T.

Proof. Consider a weighted tree $T=(V,E)$ with the *w-core* C. Let T be rooted at $c \in C$ and let $S_1, S_2, \ldots, S_k$ be the components of $T-\{c\}$. Let C extend from c into two leaves $e_1 \in S_1$ and $e_2 \in S_2$ such that *w-reduction(c,e1)* $\geq$ *w-reduction(c,e2)*. The following obviously follows:

1. For every vertex $v \in S_1$, *w-reduction(c,v)* $\leq$ *w-reduction(c,e1)*.

2. For every vertex $v \in S_2$, *w-reduction(c,v)* $\leq$ *w-reduction(c,e$_2$)*.

3. For every vertex $v \in (S_3 \cup S_4 \cup \ldots \cup S_k)$, *w-reduction(c,v)* $\leq$ *w-reduction(c,e$_2$)* $\leq$ *w-reduction(c,e$_1$)*.

Let $r \in V$ be any vertex in T. If T is re-rooted at r, then let *max-wr* = max($\{$*w-reduction(r,v)* $|v \in T\}$). Re-rooting T at r will result in the following:

1. If $r \in S_1$, then the value of *max-wr* is equal to either *w-reduction(r,e$_1$)* or *w-reduction(r,e$_2$)*.

2. If $r \in S_2$, then the value of *max-wr* is equal to *w-reduction(r,e$_1$)*.

3. If $r \in (S_3 \cup S_4 \cup \ldots \cup S_k)$, then the value of *max-wr* is equal to *w-reduction(r,e$_1$)*.

Clearly, whatever the root is *max-wr* = *w-reduction(r,e)*, where *e* is an endpoint of a *w-core* of T.

Remark 5. Let T be a tree rooted at r. If *w-reduction(r,v)* is maximum then v is an *endpoint* of the *w-core*.

3. Parallel Algorithm To Find The Endpoint Of A *W-core*

Now the formal parallel algorithm to find the *endpoint* of a *w-core* of a weighted tree is given.

Algorithm ENDPOINT

Input :

1. A tree T with n vertices rooted at r, represented in an array p(1:n) where p(i) is the parent of vertex i and p(r)=0.
2. Array w(1:n), where w(i) is the weight of vertex i.
3. Array d(1:n,1:n), where d(i,j) is the distance from vertex i to j and d(i,i)=0.

Output: Vertex e as an endpoint

Begin

1. For i = 1 to n and i $\neq$ r do in parallel
   - 1.1 s(i) = p(i)
   - 1.2 stwd(i)=0
   - 1.3 tempwd(i) = d(s(i),i) * w(i)
   - 1.4 sumtempwd(i)= $\sum$ tempwd(j) ; where s(j)= i
   - 1.5 stwd(i)= stwd(i) + sumptempwd(i)
   - 1.6 s(i) = s(s(i))
   - 1.7 while (s(i)$\neq$0) do
     - 1.7.1 tempwd(i) = d(s(i),i) * w(i)
     - 1.7.2 sumtempwd(i)= $\sum$ tempwd(j) ; where s(j)=i
     - 1.7.3 stwd(i)= stwd(i) + sumptempwd(i)
     - 1.7.4 tempwd(i) = d(p(s(i)),i) * w(i)
     - 1.7.5 sumtempwd(i)= $\sum$ tempwd(j) ; where s(j)=i
     - 1.7.6 stwd(p(i))= stwd(p(i)) + sumptempwd(i)
     - 1.7.7 s(i) = s(s(i))

   end while

   1.8 if stwd(i)=0 then stwd(i)=w(i)

   end parallel
2. temprd(r)=0
3. for i= 1 to n and i $\neq$ r do in parallel
   - 3.1 s(i)=p(i)
   - 3.2 temprd(i)=d(s(i),i)*stwd(i)
   - 3.3 wrd(i)=temprd(i) + temprd(s(i))
   - 3.4 s(i)=s(s(i))
   - 3.5 while (s(i) $\neq$ r) do
     - 3.5.1 wrd(i) = wrd(i) + temprd(s(i))
     - 3.5.2 wrd(i) = wrd(i) + temprd(p(s(i)))
     - 3.5.3 s(i) = s(s(i))

   end while

   end parallel
4. Let maxrd = ARRAYMAX(wrd)
5. Set e=i, where wrd(i) = maxrd

end

Theorem 6. An *endpoint* of a *w-core* can be found in $O(\log^2 n)$ time using $O(n^2)$ processors on the *CREW PRAM*.

Proof. Let $T=(V,E)$ be a weighted tree rooted at r. For each $v \in V$, let $p(v)$ to be the parent of v. Using lemma 2 and 3, an *endpoint* of a *w-core* can be computed as follows:

- For each $v \in V$, compute the weighted distance of the sub tree rooted at v, *stwd(v)*. This is done by using the pointer jumping technique initiated from each v, and moving up to r. As it moves up, the product $d(p(v),v)*w(v)$ is added to the sum *stwd(p(v))*. Therefore at the end of this process *stwd(v)* gives the weighted distance of the sub tree rooted at v.
- For each $v \in V$, compute the *w-reduction(r,v)* denoted by *wrd(v)*. This can be done by first computing the weighted reduction form each v to its parent, $temprd(v)=d(p(v),v)*stwd(v)$. Then the pointer jumping technique is initiated from each vertex v, and moves up to the root r. As it moves up the values *temprd(v)* and *temprd(p(v))* are added to *wrd(v)*. Therefore at the end of this process *w-reduction (r,v) = wrd(v)*.
- Compute *max-wrd* = max({*w-reduction(r,v)* | $v \in V$}).
- Identify vertices that reduce the distance by a value equal to *max-wrd*.

Step1 of ENDPOINT algorithm uses the pointer jumping technique. The while loop is repeated at most *log n* times. This is because of the following reason:

First *s(i)* is assigned with *p(i)*. So, *s(i)* gives the ancestor of i at a distance 1 from i towards the root. Whenever *s(i)* is assigned with *s(s(i))*, the distance is doubled. So the whole height of the tree is traversed in *log h* time, where h is the height of the tree. But h *d"* n, and hence the while loop is repeated *O(log n)* times.

In step 1.7.2 the summation is taken over all children. This summation can be implemented in parallel in *O(log n)* time using *O(n)* processors. The same complexity is applied to step 1.7.5 also. Hence step 1 can be implemented in $O(\log^2 n)$ time using $O(n^2)$ processors.

Step 2 and step 5 need *O(1)* time. Step 3 also uses pointer jumping technique and needs *O(log n)* time using *O(n)* processors. Step 4 clearly needs *O(log n)* time using *n/2* processors to find the maximum. Hence an *endpoint* of a *w-core* can be found in $O(\log^2 n)$ time using $O(n^2)$ processors on *CREW PRAM*.

4. Parallel Algorithm To Find The Weighted Core

The formal algorithm to find the *w-core* of a weighted tree is given below.

Algorithm W-CORE

Input : A tree T with n vertices rooted at r, represented in an array p(1:n) where p(i) is the parent of vertex i and p(r)=0.

Output: w-core of the weighted tree.

Begin

1. Call ENDPOINT algorithm to find an endpoint e1 for the tree T.
2. Call RE-ROOT algorithm to re-root T at e1.
3. Call ENDPOINT algorithm to find an endpoint e2 for the tree rooted at e1.
4. The path from e1 to e2 represents the w-core.

End

Theorem 7: A *w-core* of a weighted tree can be identified in $O(\log^2 n)$ time using $O(n^2)$ processors on the *CREW PRAM*.

Proof. Let T be a weighted tree. Using Lemmas 2, 3, 4 and Theorems 1 and 6 a *w-core* of T can be identified as follows.

1. Identify an endpoint of a w-core, e_1, using the steps given by Theorem 6.
2. Re-root the tree at e_1, using RE-ROOT algorithm given by theorem1.
3. Identify the other endpoint, e_2, as it was done in step 1.
4. The path between e_1 and e_2 forms a w-core of T. At this stage e_1 is the root and e_2 is a leaf of T. In order to identify the vertices in the path forming the w-core of T, we use the parent array p and the pointer jumping

technique initiated from e_2. As it moves up to r, all those vertices visited are in w-core.

By Theorem 6, steps 1 and 3 can be done in $O(\log^2 n)$ time using $O(n^2)$ processors. As per theorem 1 step 2 can be done in O(1) time using O(n) processors. Obviously step 4 can be done in O(log n) time using O(n) processors. Hence a *w-core* of a weighted tree can be identified in $O(\log^2 n)$ time using $O(n^2)$ processors on the *CREW PRAM*.

5. Illustration

Finding the *w-core* of a weighted tree consists of the following four steps.

1. Find an endpoint of a core of the given tree; call it e_1.
2. Re-root the tree at e_1.
3. Find another endpoint of a core of the new tree; call it e_2.
4. The path between e_1 and e_2 forms the *w-core* of the tree.

These steps are explained with the help of the tree given in figure 1.1.

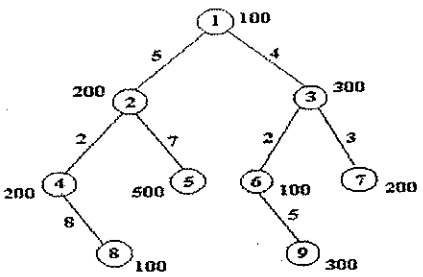

Figure 1.1 A weighted tree

Step 1: **Find *endpoint* e_1.**

a) Compute *stwd(v)*, $v \in V(T)$

v	1	2	3	4	5	6	7	8	9
Stwd(v)	-	4900	2900	800	500	1500	200	100	300

b) Compute *w-reduction(r,v)*, where r=1

v	1	2	3	4	5	6	7	8	9
wrd(v)	-	24500	11600	26100	28000	14600	12200	26900	16100

c) Since the maximum *w-reduction* is for the vertex 5, the *endpoint*, e_1=5

Step 2: Re-root T at vertex e_1=5.

The tree in figure 1.1 re-rooted at vertex 5 is given in figure 1.2

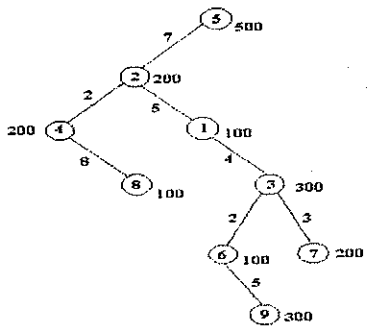

Figure 1.2 Weighted tree re-rooted at 5

Step 3: Find *endpoint* e_2.

a) Compute *stwd(v)*, $v \in V(T)$

v	1	2	3	4	5	6	7	8	9
Stwd(v)	6500	12900	2900	800	-	1500	200	100	300

b) Compute *w-reduction(r,v)*, where r=5

v	1	2	3	4	5	6	7	8	9
wrd(v)	90800	90300	92000	91900	-	92200	92600	92700	93700

c) Since the maximum *w-reduction* is for the vertex 9, the *endpoint* e_2=9

Step 4: Find *w-core*.

The path between e_1 and e_2 forms the *w-core*. As given in figure 1.3 the path from vertex 5 to vertex 9 is the *w-core* of the weighted tree. Hence the *w-core* is the path (5,2,1,3,

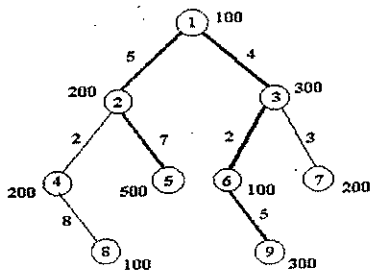

Figure 1.3 Weighted Core (5,2,1,3,6,9)

6. Conclusion

In this paper a new central structure called weighted core has been introduced and a parallel algorithm developed.

Similar concept can be extended and new weighted central structures like weighted spine, weighted path center can be defined and parallel algorithms can be developed.

References

[1] E.A. Albacea, *"Parallel algorithm for finding a core of a tree network"*, Information processing letters, 51(1994), 223-226.

[2] F. Harary, *"Graph Theory"*, Addison Wesley, Reading, Mass. (1972).

[3] E. Minieka and N.H. Patel, *"On finding the core of a tree with a specific length"*, Journal of Algorithms, 4(1983), 345-352.

[4] C.A. Morgan and P.J. Slater, *"A linear algorithm for a core of a tree"*, Journal of Algorithms, 1(1980), 247-258.

[5] K.R. Parthasarathy, *"Basic Graph Theory"*, Tata McGraw-Hill (1994).

[6] P.J. Slater, *"centrality of paths and vertices in a graph : cores and pits"*, Fourth international conference on theory and applications of graphs, Western michigan university, Weley New York (1980), 529-542.

Author's Biography :

S. P. Victor earned his M.C.A. degree from Bharathidasan University, Tiruchirappalli, and Ph.D. degree in Computer Science from M. S. University, Tirunelveli. His Ph.D. thesis was on parallel algorithms. He is currently working as Reader in the department of computer science, St. Xavier's college (Autonomous), Palayamkottai, Tirunelveli. He has published research papers on parallel algorithms in international, national journals and conference proceedings. He was the organizing secretary of the National Conference on Emerging Trends in Algorithms held at Tirunelveli in March 2006.

C. Xavier is with HCL working on Telecom Bisiness support systems for the past 6 years. Before joining HCL, he was heading the department of computer science in St.Xavier's College(Autonomous), Tirunelveli. He has written books on Web Technologies such as HTML, JavaScript, and JSP, Programming languages such as Java 2, FORTRAN, BASIC and C and Computer concepts such as Introduction to Computers, Numerical Methods and Parallel Algorithms.

He was awarded Ph.D. degree in Computer Science for his research in Parallel Algorithms. He has successfully guided five researchers for the award of Ph.D. in Computer Science. John Wiley Inc., New York, has published his book on Introduction to Parallel Algorithms in America and the same has been translated into Chinese language and published in China. He has served as a member of the Information Technology Task Force Sub-committee of Government of Tamilnadu for three years. He is a Life member of Computer Society of India and the Project Management Professional (PMP) certified by the Project Management Institute, USA.

Dr. S. Arumugam, Senior Professor (Research), Arulmigu Kalasalingam College of Engineering has 38 years of experience in teaching and research. His area of specialization is Graph Theory and its Applications. He has around 70 publications in National and International Journals. He has guided 19 candidates for Ph.D. and has served as Co-guide for 10 candidates. He is the founder editor-in-chief of AKCE International Journal of Graphs and Combinatorics.